边缘计算网络中面向负载均衡的调度机制 *

董 谦[1, 2, 3]，马宇翔[1, 2]，李 俊[1†]
(1. 中国科学院 计算机网络信息中心，北京 100190; 2. 中国科学院大学，北京 100049; 3. 佛山科学技术学院，广东佛山 528000)

摘 要：在边缘计算的应用场景中，资源的部署和分配是重要问题。针对边缘计算网络中的负载均衡需求，提出一种基于集中控制的调度机制。首先决定在哪些网络节点部署边缘计算功能，再针对用户的数据和请求，在满足相关负载均衡约束的前提下通过调度尽量降低流量的平均端到端延迟。评估结果表明，边缘计算节点的数量、计算资源和网络资源的负载均衡程度均可能影响流量的平均端到端延迟。只需选择少量合适的节点作为边缘计算节点，再将计算资源和网络资源的负载均衡调配到合适程度即可有效降低平均端到端延迟。
关键词：边缘计算；集中控制；负载均衡；分段路由
中图分类号：TP393 doi: 10.19734/j.issn.1001-3695.2018.10.0673

Load balancing oriented scheduling scheme in edge computing network

Dong Qian[1, 2, 3], Ma Yuxiang[1, 2], Li Jun[1†]
(1. *Computer Network Information Center, Chinese Academy of Sciences, Beijing* 100190, *China*; 2. *University of Chinese Academy of Sciences, Beijing* 100049, *China*; 3. *Foshan University, Foshan* Guangdong 528000, *China*)

Abstract: In the application scenario of edge computing, resource deployment and allocation are important issues. In order to address the load balancing needs in edge computing network, this paper propose a scheduling scheme based on centralized control. This scheme decides which nodes to deploy edge computing module firstly, then considering users' data and requests, minimizes the average end-to-end delay of traffic through scheduling while meeting the load balancing constraints. The evaluation results indicate that the number of edge computing nodes and the load balancing level of computing and network resources may affect the average end-to-end delay of traffic. Selecting a small number of suitable nodes as edge computing nodes and making the load balancing level of computing and network resources appropriate can effectively reduce the average end-to-end delay.
Key words: edge computing; centralized control; load balancing; segment routing

0 引言

随着用户需求的日益多样化和网络流量的不断增长，边缘计算作为云计算及 5G 的一项关键技术将发挥重要作用。边缘计算是为了满足云计算及物联网、移动网络场景的需求，在终端以及云数据中心之间的网络节点部署计算功能，能降低数据中心的负担，节约能耗，还能提高数据处理的实时性[1]。边缘计算网络结构示意图如图 1 所示。

其中，网络主要划分为接入层和核心层[2]；接入层用于连接用户设备，部署边缘计算节点，在图 1 中以云表示，且图 1 中省略了接入层网络节点之间、接入层网络节点与用户设备之间、用户设备之间（如果有）的互联链路；核心层用于连接接入层以及数据中心，在图 1 中以椭圆表示，实线表示核心层网络节点与接入层网络节点以及数据中心之间的互联链路。每个边缘计算节点都有计算、存储等能力；通常，来自用户的数据和请求经过边缘计算节点预处理后，需要转发到数据中心的流量相对来说会少得多[1]。

因此，在哪些网络节点部署边缘计算功能，如何在均衡这些节点的负载的前提下合理调度用户的数据和请求，便成为边缘计算网络所需考虑的重要问题。软件定义网络主要使用集中控制机制，具有全局视角，具备控制网络节点动作的能力。软件定义网络即集中控制机制用于边缘计算网络有突出优势[3]，本文基于集中控制提出了一种边缘计算网络中面向负载均衡的调度机制，具体来说，本文的主要贡献如下：

a)分析了边缘计算网络中的调度模型，提出了一种基于集中控制、面向负载均衡的调度机制；

b)提出了方法以决定具体在哪些网络节点部署边缘计算功能；

c)提出了方法以决定如何调度用户的数据和请求，在满足相关负载均衡约束的前提下尽量降低流量的平均端到端延迟。

1 相关工作

近年来，边缘计算得到了广泛关注，越来越多的研究致力于解决边缘计算在移动网络、物联网、车联网等领域的应用，以及边缘计算中计算、网络等资源的调度问题[4-6]。软件定义网络、分段路由（segment routing）等新兴网络技术逐渐应用于边缘计算领域[3,7,8]。

Shi 等人[1]总结了边缘计算的概念和原理，介绍了有代表性的应用实例，并展望了存在的主要挑战。Mao 等人[9]主要

收稿日期：2018-10-10；**修回日期：**2018-11-28 **基金项目：**国家重点研发计划资助项目（2017YFB1401500）；国家自然科学基金资助项目（61672490）；中国科技云建设工程（Y72923）

作者简介：董谦（1986-），男，湖北咸宁人，讲师，博士研究生，主要研究方向为未来互联网、软件定义网络等；马宇翔（1991-），男，博士研究生，主要研究方向为网络体系结构、网络安全等；李俊（1968-），男（通信作者），研究员，博士，主要研究方向为未来互联网、网络安全等（lijun@cnic.cn）.

关注移动边缘计算，讨论了系统部署、缓存机制、移动性管理、节能和隐私等方面的研究进展和挑战，还介绍了标准化方面的努力以及一些典型的应用场景。Mach 等人[10]主要关注移动边缘计算中的计算卸载（offloading）用例，重点总结了完全卸载和部分卸载这两种情况的研究现状；完全卸载的主要目标是最小化执行延迟（delay），或是在满足执行延迟约束的情况下最小化能耗，或是取得执行延迟与能耗间的平衡；部分卸载的主要目标是在满足执行延迟约束的情况下最小化能耗，或是取得执行延迟与能耗间的平衡；考虑如何进行合适的计算资源分配，并区分将计算分配到单个节点和分配到多个节点这两种类别。Sun 等人[11]提出 PRIMAL，考虑用户与计算节点的端到端延迟，以及计算和存储资源的迁移成本，以取得端到端延迟与迁移成本间的平衡。Wang 等人[12]研究了边缘计算中的负载放置问题，以决定如何在边缘计算节点上放置应用。以上工作表明，在边缘计算的应用场景中，资源的部署和分配是重要问题，用户设备到边缘计算节点的端到端延迟是主要优化目标之一；然而，较少有工作关注计算资源和网络资源的负载均衡需求对端到端延迟的影响。

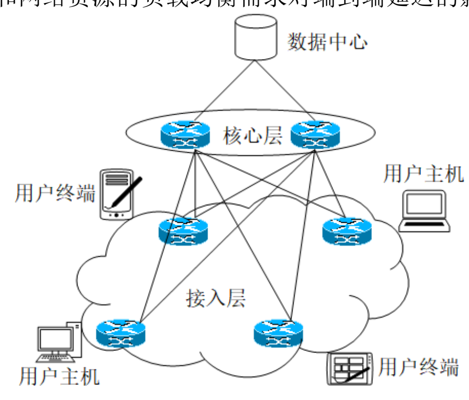


图 1　边缘计算网络结构示意图

Fig. 1　The structure of edge computing network

另一方面，Baktir 等人[3]提出将软件定义网络与边缘计算相结合，由控制器来管理数据流，完成服务编排和其他管理任务，通常包含服务发现、服务调试与迁移、性能调优与优化、用户切换等模块；其中，性能调优和优化模块主要关注网络和计算资源的使用情况，管理边缘计算节点上的负载。Filsfils 等人[13]提出分段路由，可在灵活配置数据流转发路径的同时减轻控制器下发转发规则的负担；Hartert 等人[14]提出集中式优化器 DEFO，DEFO 用于控制转发路径，而分段路由的灵活性以及可扩展性在其中发挥了重要作用；Desmouceaux 等人[15]提出 6LB，采用基于 IPv6 的分段路由技术引导数据包，设计负载均衡器。以上工作表明，集中控制机制与边缘计算相结合能优化负载均衡与网络性能；分段路由则支持对路径的灵活控制，增强了控制器的可扩展性。

因此，本文将负载均衡需求视作约束，在满足这些约束的前提下，基于集中控制和分段路由技术，通过调度尽量降低流量的平均端到端延迟。

2　整体架构和问题分析

基于对实际场景的抽象，结合本机制的设计思路，设图 1 中核心层以下的所有网络节点都支持分段路由技术，分段路由转发规则只需在源节点配置，增强了集中控制器的可扩展性；再将这些网络节点分为以下三类：

a)用户设备，除具备网络节点的基本功能外，还向边缘计算节点发送数据和请求；

b)边缘计算节点，除具备网络节点的基本功能外，还处理用户设备发送的数据和请求，具备计算和存储等资源，处理完毕后，将处理后的数据和对请求的响应返回到用户设备，或是发送到数据中心；

c)既非用户设备也无边缘计算能力的节点，即中间节点，只具备网络节点的基本功能。

本机制中除了网络节点外还有集中控制器，以一个常见的网络拓扑为例，包括 6 个用户设备、3 个边缘计算节点和 3 个中间节点，整体架构和控制器功能示意图如图 2 所示。控制器只需接入边缘计算网络即可使用 NETCONF[16]等南向接口协议获取网络和各类资源的使用情况，配置设备，图 2 中未将此功能表示出来；带单向箭头的实线表示控制器根据要求选择合适的网络节点部署边缘计算功能；带箭头的虚线表示控制器接收用户设备的调度需求，下发分段路由转发规则给用户设备及边缘计算节点；网络节点间的实线表示它们之间的互联链路。

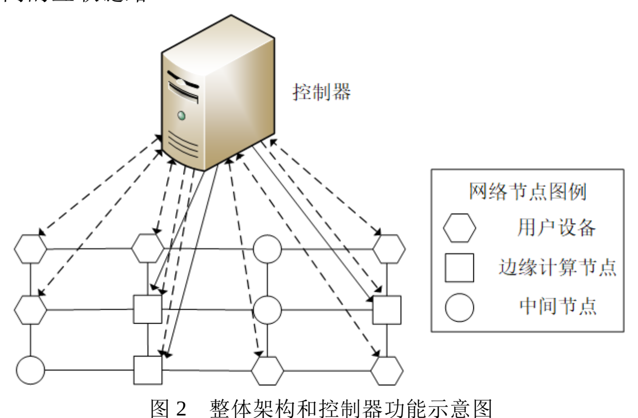


图 2　整体架构和控制器功能示意图

Fig. 2　System architecture and the controller’s function

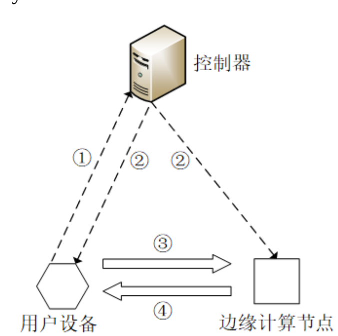


图 3　用户设备的调度需求处理步骤示意图

Fig. 3　Processing steps of user equipment’s scheduling demands

当边缘计算节点确定后，一旦用户设备有数据和请求产生，处理步骤如下：a)用户设备将数据和请求的流量信息发送给控制器；b)控制器根据当前网络和各类资源的使用情况，计算得出应将这些流量转发到哪些边缘计算节点，应通过哪些路径转发，将分段路由转发规则下发给用户设备，通知相关的边缘计算节点预留资源；如果处理后的数据和对请求的响应还需要发送给用户设备，控制器同样应根据当前网络和各类资源的使用情况，计算得出应通过哪些路径转发，将分段路由转发规则下发给边缘计算节点；c)用户设备收到控制器的响应后，按照控制器下发的规则发送流量；d)边缘计算节点接收流量后进行处理，收到控制器的响应后，按照控制器下发的规则发送流量，最终用户设备收到边缘计算节点发送的处理后的数据和对请求的响应。用户设备的调度需求处理步骤示意图如图 3 所示。

其中，用户设备与控制器间带单向箭头的虚线对应图 2 中带双向箭头的虚线，表示用户设备发送调度需求，控制器

向用户设备下发转发规则；控制器与边缘计算节点间带单向箭头的虚线对应图 2 中带单向箭头的虚线，表示控制器向边缘计算节点下发转发规则；单向粗箭头表示用户设备与边缘计算节点间互相发送流量；图 3 中省略了中间节点，因其支持分段路由技术，故控制器无须向其下发转发规则。

网络拓扑通常表示为有向图，网络节点用 i 表示，所有核心层以下的网络节点的集合用 N 表示，链路是有向的，用 e 表示，e 的集合用 E 表示；任意某个 i 只属于上述三类网络节点中的某一类，为便于说明节点所属的类别，可将用户设备、边缘计算节点分别再用 s, t 表示，s, t 的集合分别用 S, T 表示；设 t 上计算资源的处理能力为 v_t，为方便计算，其单位与用户设备的发送速率一致，都为 bps。设网络拓扑信息已知，S 也已知，T 通过 3.1 节中的方法得出；将 T 中元素的个数用 $|T|$ 表示。

为简化讨论，设某个 s 只有某个应用（容易扩展到多个应用并存的情况）产生的待发送流量，其大小即发送速率为 d_s，且 $d_s>0$，其中发送给某个 t 的部分大小为 $d_s(t)$，最大计算资源利用率设置为 λ，且 $0\leqslant\lambda\leqslant1$，有

$$\begin{cases} d_s(t)\ge 0 & \text{当允许将计算分配到多个节点时，} \\ d_s(t)\in\{0,d_s\} & \text{当只允许将计算分配到单个节点时，} \end{cases} \quad \forall s\in S, t\in T \tag{1}$$

$$d_s=\sum_{t\in T} d_s(t), \forall s\in S \tag{2}$$

$$\sum_{s\in S} d_s(t)\le \lambda v_t, \forall t\in T \tag{3}$$

$$\sum_{s\in S} d_s \le v_t, \forall t\in T \tag{4}$$

式(1)表明了当允许将计算分配到多个节点时及只允许将计算分配到单个节点时 $d_s(t)$ 的取值要求；式(2)表明 d_s 全部发送；式(3)表明每个 t 接收的流量能被其实际允许使用的计算资源处理，显然，在满足式(1)(2)的前提下，λ 越小则表明所有 t 上的计算资源利用率可能的范围越小，故本文用 λ 来衡量所允许的计算资源的负载均衡程度；式(4)表明即使只有一个边缘计算节点，当 λ 为 1 时也能处理所有 s 的待发送流量，即 $|T|$ 最小可为 1，保证了网络的健壮性。

另外，t 接收 $d_s(t)$ 并处理后，再返回 $r_t(s)$ 到 s，为简化讨论，本文设所有 s 的待发送流量的类型相同（容易扩展到多种类型并存的情况），s 的待发送流量的类型决定了系数 α，$\alpha\geqslant0$，使

$$r_t(s)=\alpha d_s(t), \forall s\in S, t\in T \tag{5}$$

一般来说，α 较小乃至为 0 表示 s 的待发送流量主要是需要处理的数据；α 较大表示 s 的待发送流量主要是需要处理的请求。

若已知网络拓扑信息则容易根据预设条件及分段路由的转发规则预先计算两节点间无环路的候选转发路径。将某条候选转发路径表示为 p，用 p_e 表示链路 e 是否在路径 p 上，若在则 p_e 为 1，否则为 0；考虑 s 到 $i\in N\backslash S$ 和 $i\in N\backslash S$ 到 s 的候选转发路径集合，分别用 P_{si} 和 P_{is} 表示，"\" 是相对补集符号；用 p 的跳数 h_p 衡量流量转发路径采用 p 时的端到端延迟，对于每一对 s, i，P_{si} 中的路径和 P_{is} 中的路径一一对应且方向相反，故 P_{si} 和 P_{is} 中路径的跳数的最小值相同，用 h_{si} 表示 $\min_{p\in P_{si}}\{h_p\}$，$w_p$ 表示 p 的路由代价即权重，w_{si} 表示 $\min_{p\in P_{si}\text{且}h_p=h_{si}}\{w_p\}$；对于节点 $i\in T$，由于其通常再用 t 表示，相应的 P_{si}、P_{is}、h_{si}、w_{si} 也可用 P_{st}、P_{ts}、h_{st}、w_{st} 表示。再设路径 p 上承载的流量大小为 x_p，有

$$d_s(t)=\sum_{p\in P_{st}} x_p, \forall s\in S, t\in T \tag{6}$$

$$r_t(s)=\sum_{p\in P_{ts}} x_p, \forall t\in T, s\in S \tag{7}$$

式(6)(7)表明 $d_s(t)$ 及 $r_t(s)$ 已分配转发路径。值得说明的是，在本文中，只允许将计算分配到单个节点时通常还要求转发路径只用某一条，以避免出现数据包重排[17]，由式(1)~(2)(5)~(7)，有

$$x_p\in\{0,d_s\}, \forall s\in S, t\in T, p\in P_{st} \tag{8}$$

$$x_p\in\{0,\alpha d_s\}, \forall t\in T, s\in S, p\in P_{ts} \tag{9}$$

允许将计算分配到多个节点则无此限制，有

$$x_p\ge 0, \forall s\in S, t\in T, p\in P_{st}\cup P_{ts} \tag{10}$$

最后，定义流量的平均跳数 H 为

$$H=\frac{\sum_{s\in S}\sum_{t\in T}\sum_{p\in P_{st}} x_p h_p+\sum_{t\in T}\sum_{s\in S}\sum_{p\in P_{ts}} x_p h_p}{\sum_{s\in S} d_s(1+\alpha)} \tag{11}$$

由式(11)计算得到的 H 用于衡量流量的平均端到端延迟。若还考虑网络资源，以链路容量为例，链路容量用 c_e 表示，最大链路利用率设置为 θ，且 $0\leqslant\theta\leqslant1$，与 λ 类似，本文用 θ 来衡量所允许的网络资源的负载均衡程度，有

$$\sum_{s\in S}\sum_{t\in T}\sum_{p\in P_{st}} x_p p_e+\sum_{t\in T}\sum_{s\in S}\sum_{p\in P_{ts}} x_p p_e\le\theta c_e, \forall e\in E \tag{12}$$

式(12)表明实际允许使用的网络资源可能制约转发路径的选择，进而影响求得的 H。通常，在 α、λ、θ 等条件相同的情况下，求得的 H 应尽量小，此时优化任务写为 minimize H。特别的，若不考虑网络资源负载均衡约束，当优化任务为 minimize H 时，s 到 t 和 t 到 s 所选择的转发路径的 h_p 应等于 h_{st}，此时只需考虑式(1)~(3)，式(11)可化简为

$$H=\frac{\sum_{s\in S}\sum_{t\in T} d_s(t)h_{st}}{\sum_{s\in S} d_s} \tag{13}$$

式(13)表明此时 α 不影响求得的 H。

3　机制设计

3.1　边缘计算节点选择

选择边缘计算节点前，已知的信息包括网络拓扑信息，S 及相应的 d_s，所有 s 到所有 $i\in N\backslash S$ 的 h_{si}、w_{si}。选择边缘计算节点时，不考虑计算资源和网络资源负载均衡约束，优化任务为 minimize H；由于 T 未知，应先设置最后得到的 $|T|$ 等于 k。

用 u_i 表示节点 $i\in N\backslash S$ 是否被选作 t，若 i 被选作 t 则 u_i 为 1，否则为 0；设某个 s 产生的待发送流量中发送给某个 i 的部分大小为 $d_s(i)$；设置一个足够大的数 m，忽略单位只取数值，通常应使 $\min_{s\in S}\{\min_{i\in N\backslash S}\{md_s h_{si}\}\}\gg\max_{s\in S}\{\max_{i\in N\backslash S}\{d_s w_{si}\}\}$，有

$$\text{minimize}\ \sum_{s\in S}\sum_{i\in N\backslash S} d_s(i)mh_{si}+\sum_{s\in S}\sum_{i\in N\backslash S} d_s(i)w_{si} \tag{14}$$

$$0\le d_s(i)\le u_i d_s, \forall s\in S, i\in N\backslash S \tag{15}$$

$$d_s=\sum_{i\in N\backslash S} d_s(i), \forall s\in S \tag{16}$$

$$|T|=\sum_{i\in N\backslash S} u_i=k \tag{17}$$

式(14)是优化求解器进行边缘计算节点选择时的优化任务，实际意义是在 minimize H 的前提下，借由 m 来确保当有多个解使得 H 最小时再根据路由代价在这些解中选择一个解

作为最终解；式(15)由 $d_s(i)$的定义得到，还表明当 i 未被选作 t 时 $d_s(i)$为 0；式(16)由式(2)改写得到；式(17)表明最后得到的$|T|$等于 k。求解可使用 Gurobi[18]等优化求解器，优化任务为式(14)，只需考虑式(15)~(17)；求得 T 后，对于 $i\in T$，由于其通常再用 t 表示，相应的 $d_s(i)$、h_{si} 也可用 $d_s(t)$、h_{st} 表示，H 由式(13)计算得到。

也可对 S 设置待定集合 S'，设置两个临时变量 O'和 O''，再使用下列算法

算法 1　边缘计算节点选择

1) 输入网络拓扑信息，S 及相应的 d_s，k，m，所有 s 到所有 $i\in N\backslash S$ 的 h_{si}、w_{si}

2) T 初始化为空集，S'初始化为 S；

3) for $temp$ = 1 to k do

4) 　$O'\leftarrow+\infty$；

5) 　候选边缘计算节点初始化为空；

6) 　for each $i\in N\backslash(S\cup T)$ do

7) 　　$O''\leftarrow m\sum_{s\in S'}d_s h_{si}+\sum_{s\in S'}d_s w_{si}$；

8) 　　if $O''<O'$ then

9) 　　　$O'\leftarrow O''$；

10) 　　　候选边缘计算节点←i；

11) 　　else

12) 　　end if

13) 　end for

14) 　将候选边缘计算节点添加到 T；

15) 　for each $s\in S'$ do

16) 　　if $\min_{t\in T}\{h_{st}\}=\min_{i\in N\backslash S}\{h_{si}\}$ then

17) 　　　在 S'中去除 s；

18) 　　else

19) 　　end if

20) 　end for

21) 　if S'为空集 then

22) 　　Break；

23) 　else

24) 　end if

25) end for

26) $H\leftarrow\dfrac{\sum_{s\in S}d_s\min_{t\in T}\{h_{st}\}}{\sum_{s\in S}d_s}$；

27) if $|T|<k$ then

28) 　在 $N\backslash(S\cup T)$中随机选择$(k-|T|)$个节点添加到 T；

29) else

30) end if

边缘计算节点选择算法的输出为 T、H，且$|T|=k$；根据 T 及相应的 v_t 配置边缘计算节点上的计算资源。

3.2　调度计算

然后，控制器可进行调度计算。每次计算前，应设置 α、λ 和 θ 的值，通常调度计算时的条件越多则计算难度越大。

如果不考虑网络资源负载均衡约束，优化任务为 minimize H，只需考虑式(1)~(3)，H 由式(13)计算得到。求解可使用优化求解器，也可使用下列算法

算法 2　调度计算

a)输入 S 及相应的 d_s，T 及相应的 v_t，λ，m，所有 s 到所有 t 的 h_{st}、w_{st}

b)将所有 s 的 $d_s(t)$初始化为 0，对所有 s 按照 d_s 从大到小的顺序依次计算，每次求当前 s 的 $d_s(t)$时，不考虑还未计算过的 s，只考虑已计算过的 s 和当前 s，在满足式(1)~(3)的前提下使当前 s 的 $m\sum_{t\in T}d_s(t)h_{st}+\sum_{t\in T}d_s(t)w_{st}$ 最小，后续计算中保持已计算过的 s 的 $d_s(t)$不变，所有 s 计算完成后，有解则暂存为解 1；

c)将所有 s 的 $d_s(t)$初始化为 0，对所有 s 按照 d_s 从小到大的顺序依次计算，每次求当前 s 的 $d_s(t)$时，不考虑还未计算过的 s，只考虑已计算过的 s 和当前 s，在满足式(1)~(3)的前提下使当前 s 的 $m\sum_{t\in T}d_s(t)h_{st}+\sum_{t\in T}d_s(t)w_{st}$ 最小，后续计算中保持已计算过的 s 的 $d_s(t)$不变，所有 s 计算完成后，有解则暂存为解 2；

d)若第 2、3 行都有解则比较解 1 和解 2，根据式(13)分别得出它们的 H，取相对较小者作为最终解及 H，若第 2、3 行只有一个有解则直接选择这个解及 H；

调度计算算法的输出为所有 s 的 $d_s(t)$、H；控制器对每个不为 0 的 $d_s(t)$选择一条满足 $p\in P_{st}$ 及 $h_p=h_{st}$ 条件的转发路径 p，使 $x_p=d_s(t)$；由 $d_s(t)$计算得到 $r_t(s)$，同样对每个不为 0 的 $r_t(s)$选择一条满足 $p\in P_{ts}$ 及 $h_p=h_{st}$ 条件的转发路径 p，使 $x_p=r_t(s)$；根据 x_p 向用户设备及边缘计算节点下发相关配置。

如果还考虑网络资源负载均衡约束，优化任务为 minimize H，考虑式(1)~(3)(5)~(10)(12)，H 由式(11)计算得到，求解使用优化求解器，输出为所有 x_p，只根据大于 0 的 x_p 向用户设备及边缘计算节点下发相关配置。

4　实验评估与分析

本文从公开数据集 SNDLib[19]获取实验网络拓扑信息，此拓扑中共有 22 个节点，预设的 S 包含其中 10 个节点；链路容量均为 40000Mbps。再从 SNDLib 的流量矩阵（traffic matrix, TM）数据集中取出 3 个 TM，对每个 TM 分别统计这 10 个 s 的出流量之和作为各自的 d_s，由这 3 个 TM 得到的 S 及相应的 d_s 作为场景 1、2、3；将 v_t 均设为 40000Mbps。

先求得所有 s 与所有 $i\in N\backslash S$ 间的 P_{si}、P_{is}，每一对 s, i 的 P_{si}、P_{is} 中有它们间满足 SLD[20]（segment list depth）不大于 2 及无环路条件的路径[21]，继而得到所有 s 到所有 $i\in N\backslash S$ 的 h_{si}、w_{si}，用于后续计算。进行边缘计算节点选择，不考虑计算资源和网络资源负载均衡约束，对于场景 1，改变 k 的值，对比使用优化求解器求得的结果与算法 1 求得的结果如图 4 所示。

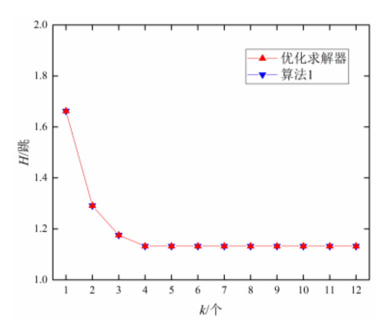


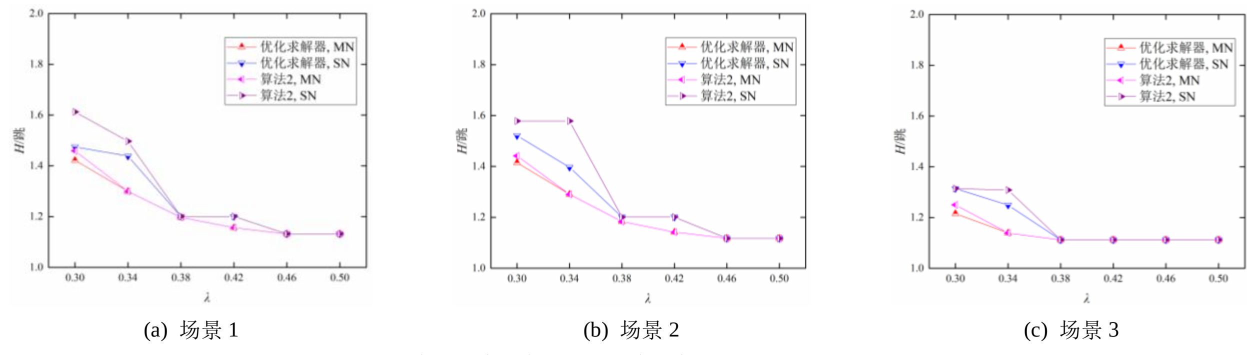
图 4　场景 1 下不同 k 与求得的 H

Fig. 4　K and corresponding H in scenario 1

图 4 表明，当 k 相同时，使用优化求解器求得的 H 与算法 1 求得的 H 相同，由于优化求解器求得的 H 是相应条件下的最小值，因此使用算法 1 求得的 H 也是相应条件下的最小值；k 小于等于 4 时，k 越大，求得的 H 越小，而 k 大于等于 4 时，求得的 H 相同，原因在于当 k 小于 4 时，有部分 s

的 $\min_{t\in T}\{h_{st}\}>\min_{i\in N\setminus S}\{h_{si}\}$，而当 k 增大到 4 以后，所有 s 的 $\min_{t\in T}\{h_{st}\}=\min_{i\in N\setminus S}\{h_{si}\}$，即使再增加 t，也不可能求得更小的 H。另外，当 k 分别取 1、2、3、4 时，使用优化求解器与算法 1 求得的 T 也相同，而算法 1 是相对更为简单的贪心算法。上述结果表明，使用算法 1 选择少量合适的节点作为边缘计算节点，不仅可有效降低平均跳数，还简化了计算过程。

然后，将 k 设置为 4，基于场景 1 由算法 1 计算得出 T，因此有所有 s 到所有 t 的 h_{st}、w_{st}；只允许将计算分配到单个节点用 SN 表示，允许将计算分配到多个节点用 MN 表示；进行调度计算，先不考虑网络资源负载均衡约束，对于场景 1、2、3，改变 λ 的值，对比使用优化求解器求得的结果与算法 2 求得的结果如图 5 所示。



(a) 场景 1　　(b) 场景 2　　(c) 场景 3

图 5　不考虑网络资源负载均衡约束时各场景下不同 λ 与求得的 H

Fig. 5　Λ and corresponding H in various scenarios without considering network resource load balancing constraints

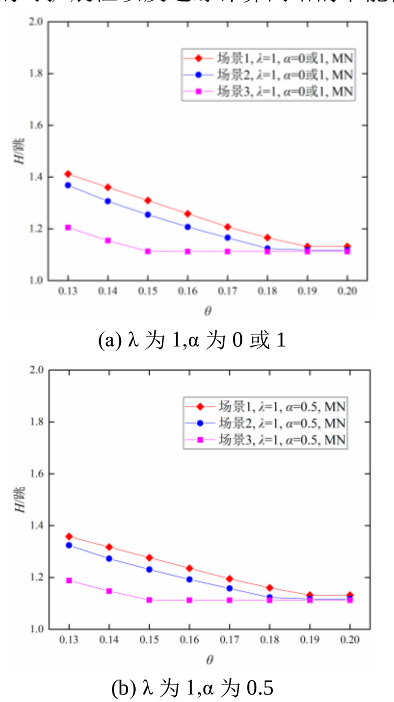
图 5 表明，若 λ 较小，允许将计算分配到多个节点时使用优化求解器求得的 H 比算法 2 求得的 H 略小，只允许将计算分配到单个节点时使用优化求解器求得的 H 比算法 2 求得的 H 更小或与之相同；若 λ 较大，使用优化求解器求得的 H 与算法 2 求得的 H 相同；通常，在 λ 增加到一定程度以前，当 λ 相同时，允许将计算分配到多个节点时求得的 H 小于等于只允许将计算分配到单个节点时求得的 H；当 λ 增加到一定程度后，λ 继续增加也不会改变此时求得的 H，这说明所有 s 的 $\min_{t\in T且d_s(t)>0}\{h_{st}\}=\max_{t\in T且d_s(t)>0}\{h_{st}\}=\min_{t\in T}\{h_{st}\}$。上述结果表明，所允许的计算资源的负载均衡程度对平均跳数可能有较大影响，特别是只允许将计算分配到单个节点时，λ 越小则平均跳数可能会有较大增加，而允许将计算分配到多个节点则平均跳数可能增加的幅度相对较小；当 λ 较大时，算法 2 也能取得较好效果。

最后，仍将 k 设置为 4，基于场景 1 由算法 1 计算得出 T，因此也有所有 s 与所有 t 间的 P_{st}、P_{ts}；只考虑允许将计算分配到多个节点的情况，且取 λ 为 1；进行调度计算，考虑网络资源负载均衡约束，分别取 α 为 0、0.5、1，对于场景 1、2、3，改变 θ 的值，使用优化求解器求得的结果如图 6 所示。

图 6 表明，在 θ 增加到一定程度以前，θ 越小，求得的 H 越大；当 θ 增加到一定程度后，θ 继续增加也不会改变此时求得的 H；当 θ 相同时，α 为 0 或 1 时求得的 H 相同，这是由于 s 发送和接收的流量方向相反，接收流量大小为 0 或等于发送流量大小则求得的 H 实际上只取决于发送流量；在 θ 增加到一定程度以前且当 θ 相同时，α 为 0 或 1 时求得的 H 大于 α 为 0.5 时求得的 H，这是由于 s 发送和接收的流量不相等且接收的流量不为 0 时，两者中较大者受网络资源负载均衡约束的限制相对较小者有更大比例的流量使用跳数更多的路径转发。上述结果表明，所允许的网络资源的负载均衡程度也有可能影响平均跳数，θ 越小则平均跳数可能增加。

5　结束语

本文针对边缘计算网络中的负载均衡需求，提出了一种基于集中控制的调度机制，在满足相关负载均衡约束的前提下通过调度尽量降低流量的平均端到端延迟，基于模型进行分析，提出了算法以决定具体在哪些网络节点部署边缘计算功能和如何调度用户的数据和请求。评估结果表明，边缘计算节点的数量、计算资源和网络资源的负载均衡程度均可能影响流量的平均端到端延迟；只需选择少量合适的节点作为边缘计算节点，再将计算资源和网络资源的负载均衡调配到合适程度，即可有效降低平均端到端延迟。下一步工作计划在已有基础上，考虑如何在链路、风险共享链路组（shared risk link group）、节点等出现故障的情况进行合理调度，且兼顾集中控制器的可扩展性以及边缘计算网络的节能需求。



(a) λ 为 1,α 为 0 或 1

(b) λ 为 1,α 为 0.5

图 6　各场景下不同 θ 与求得的 H

Fig. 6　Θ and corresponding H in various scenarios

参考文献：

[1] Shi Weisong, Cao Jie, Zhang Quan, *et al*. Edge computing: vision and challenges [J]. IEEE Internet of Things Journal, 2016, 3 (5): 637-646.

[2] Byers C C. Architectural imperatives for fog computing: use cases, requirements, and architectural techniques for FOG-enabled IoT networks [J]. IEEE Communications Magazine, 2017, 55 (8): 14-20.

[3] Baktir A C, Ozgovde A, Ersoy C. How can edge computing benefit from software-defined networking: a survey, use cases, and future directions [J]. IEEE Communications Surveys & Tutorials, 2017, 19 (4): 2359-2391.

[4] Zhao Zhiwei, Min Geyong, Gao Weifeng, *et al*. Deploying edge computing nodes for large-scale IoT: a diversity aware approach [J]. IEEE Internet of Things Journal, 2018, 5 (5): 3606-3614.

[5] Zuo Yuan, Wu Yulei, Min Geyong, *et al.* Learning-based network path planning for traffic engineering [J]. Future Generation Computer Systems, 2019, 92: 59-67.

[6] Ma Yuxiang, Wu Yulei, Ge Jingguo, *et al.* An architecture for accountable anonymous access in the Internet-of-Things network [J]. IEEE Access, 2018, 6: 14451-14461.

[7] Cheng Xiangle, Wu Yulei, Min Geyong, *et al.* Network function virtualization in dynamic networks: a stochastic perspective [J]. IEEE Journal on Selected Areas in Communications, 2018.

[8] Li Jianhua, Jin Jiong, Yuan Dong, *et al.* EHOPES: data-centered fog platform for smart living [C]// Proc of IEEE International Telecommunication Networks and Applications Conference. 2015: 308-313.

[9] Mao Yuyi, You Changsheng, Zhang Jun, *et al.* A survey on mobile edge computing: the communication perspective [J]. IEEE Communications Surveys & Tutorials, 2017, 19 (4): 2322-2358.

[10] Mach P, Becvar Z. Mobile edge computing: a survey on architecture and computation offloading [J]. IEEE Communications Surveys & Tutorials, 2017, 19 (3): 1628-1656.

[11] Sun Xiang, Ansari N. PRIMAL: profit maximization avatar placement for mobile edge computing [C]// Proc of IEEE International Conference on Communications. 2016: 1-6.

[12] Wang Shiqiang, Zafer M, Leung K K. Online placement of multi-component applications in edge computing environments [J]. IEEE Access, 2017, 5: 2514-2533.

[13] Filsfils C, Nainar N K, Pignataro C, *et al.* The segment routing architecture [C]// Proc of IEEE Global Communications Conference. 2015: 1-6.

[14] Hartert R, Vissicchio S, Schaus P, *et al.* A declarative and expressive approach to control forwarding paths in carrier-grade networks [J]. ACM SIGCOMM Computer Communication Review, 2015, 45 (4): 15-28.

[15] Desmouceaux Y, Pfister P, Tollet J, *et al.* 6LB: scalable and application-aware load balancing with segment routing [J]. IEEE/ACM Transactions on Networking, 2018, 26 (2): 819-834

[16] Enns R, Bjorklund M, Schoenwaelder J, *et al.* RFC 6241, Network configuration protocol (NETCONF) [S]. IETF 2011.

[17] Kandula S, Katabi D, Sinha S, *et al.* Dynamic load balancing without packet reordering [J]. ACM SIGCOMM Computer Communication Review, 2007, 37 (2): 51-62.

[18] Gurobi Optimization, LLC. Gurobi optimizer reference manual [EB/OL]. [2018-11-23]. http://www. gurobi. com.

[19] Orlowski S, Wessäly R, Pióro M, *et al.* SNDlib 1. 0-survivable network design library [J]. Networks, 2010, 55 (3): 276-286.

[20] Moreno E, Beghelli A, Cugini F. Traffic engineering in segment routing networks [J]. Computer Networks, 2017, 114: 23-31.

[21] 董谦，李俊，马宇翔，等. 软件定义网络中基于分段路由的流量调度方法 [J]. 通信学报，2018 39(11): 23-35. (Dong Qian, Li Jun, Ma Yuxiang, *et al.* Traffic scheduling method based on segment routing in software-defined networking [J]. Journal on Communications, 2018 39 (11): 23-35.)